IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	Cary L. Bates, et al.	:	Date: May 1, 2007
Group Art Unit:	2191	:	IBM Corporation
Examiner:	Z. Wei	:	Intellectual Property Law
Serial No.:	10/720,961	:	Dept. 917, Bldg. 006-1
Filed:	November 24, 2003	:	3605 Highway 52 North
Title:	METHOD AND APPARATUS FOR EFFICIENTLY DEVELOPING ENCODED INSTRUCTIONS BY TRACKING MULTIPLE UNVERIFIED INSTANCES OF REPETITIVE CODE SEGMENTS	:	Rochester, MN 55901

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF IN SUPPORT OF APPEAL

FROM THE PRIMARY EXAMINER TO THE BOARD OF APPEALS

Sir:

This is an appeal of a Final Rejection under 35 U.S.C. §102(b) and 35 U.S.C. §103(a) of claims 1, 3-8, 10-13 and 15-18 of Application Serial No. 10/720,961, filed November 24, 2003. This brief is submitted pursuant to a Notice of Appeal filed March 20, 2007, as required by 37 C.F.R. §1.192.

1. Real Party in Interest

International Business Machines Corporation of Armonk, NY, is the real party in interest. The inventors assigned their interest as recorded on March 3, 2004, on Reel 014391, Frame 0457.

2. Related Appeals and Interferences

There are no related appeals nor interferences pending with this application.

3. Status of Claims

Claims 1, 3-8, 10-13, and 15-18 are pending and stand finally rejected. The claims on appeal are set forth in the Appendix of Claims

4. Status of Amendments

No amendments were submitted following Final Rejection.

5. Summary of Claimed Subject Matter

The invention herein relates to developing computer programming source code, particularly where source code is edited by copying sequences of code statements from one code location to another, as by interactive cutting and pasting. Independent claim 1 recites a method, and independent claim 13 recites an analogous computer program product, wherein copies are automatically tracked, and subsequent changes to a source code statement are automatically propagated to multiple copies of the statement. Independent claim 7 recites a method wherein a source code statement is checked for verification status before copying.

In accordance with the preferred embodiment, a source editor in a code development environment supports the generation and editing of source code by a code

developer [Spec. p. 9, lines 21-24; p. 11.lines 5- 15; p. 14. lines 12- 20; p. 16, lines 8 - p. 17, line 17; Fig. 4, step 401; Fig. 5]. The code development environment automatically tracks a status of each source statement, the status preferably including whether multiple copies of the source code statement exist (claims 1 and 13) and whether the statement has been verified (claim 7) [Spec. p. 5, lines 2-4; p. 11, lines 16-26; p. 12; line 13 - p. 13, lines 19; p. 14, lines 16-18; p. 14, line 24 - p. 15, line 4; p. 16 lines 14-18 and 25-26; Fig. 3; Fig. 4 steps 402, 405; Fig. 5 steps 501, 506].

In accordance with an aspect claimed in claims 1 and 13, if a source code statement is edited by the developer, a development environment mechanism automatically determines whether multiple copies of the source code statement exist using the status information, and can automatically propagate the changes to each of the multiple copies [Spec. p. 5, lines 5-8; p. 17, lines p. 19, line 16; Fig. 6] .

In accordance with an aspect claimed in claim 7, if a user command is received to copy a source code statement to another location, a development environment mechanism automatically determines whether the statement to be copied has been verified using the status information, and takes appropriate action (such as a warning message) if not. [Spec. p. 5, lines 9-12; p. 20, line1 - p. 21, line 8; Fig. 7].

6. Grounds of Rejection To Be Reviewed on Appeal

Claims 1, 3-8 and 10-12 are finally rejected under 35 U.S.C. §102(b) as anticipated by Robert C. Miller, "Lightweight Structure in Text" (Ph.D. Thesis, Carnegie Mellon University, May 2002) (herein referred to as *Miller*). Claims 13 and 15-18 are finally rejected under 35 U.S.C. §103(a) as obvious over *Miller*. The only issues in this appeal are whether the claims are either anticipated by *Miller*, or prima facie obvious over *Miller*.

7. Argument

Appellants contend that the Examiner failed to establish adequate grounds of rejection for the following reasons:

I. The Examiner improperly rejected independent claim1 (and dependent claims 3-6) under 35 U.S.C. §102(b) because *Miller* does not disclose key claim limitations, specifically automatically maintaining status of source code statements and using the status to identify copies for propagating changes [page 7 below].

II. The Examiner improperly rejected independent claim 7 (and dependent claims 8 and 10-12) under 35 U.S.C. §102(b) because *Miller* does not disclose key claim limitations, specifically automatically maintaining status of source code and using the status to identify unverified source code statements [page 12 below].

III. The Examiner improperly rejected the claims under 35 U.S.C. §103(a) because *Miller* is directed to entirely different subject matter, does not teach or suggest appellant's key features, and only with the added benefit of hindsight can key claim limitations be extracted from *Miller* [page 15 below].

Overview of Invention

A brief overview of appellants' invention in light of existing art will be helpful in appreciating the issues herein. Appellants' invention is intended to provide an improved user interface for developing computer programming source code. As is well known, source code is typically generated and edited with an interactive source code editor. Such interactive source code editors often have the capability to "cut and paste" or otherwise copy a single source code statement (or sequence of statements) from one location to another location within the source code. Since source code often involves many repetitive tasks, developers frequently use such cut and paste capability to copy statements to different parts of the source code. The copied statements may be subsequently edited, although they are often copied exactly and left as is in the new location.

When an original source code statement contains an error, and is copied to one or more other locations in the code, the error is of course multiplied. Conventionally, when the error is subsequently discovered, the developer must manually correct each of the multiple instances of the error. Alternatively, the developer could manually correct one instance of the error, and then manually copy the corrected code to each of the other multiple error instances. Either way, the conventional process involves considerable manual intervention, and is itself prone to additional error. The developer may neglect to identify all of the multiple instances of the error; the developer may make another error when correcting one of the multiple instances; etc.

Appellants propose to improve this process by use of two techniques, which are claimed separately herein. In either technique, a development environment mechanism

automatically maintains status information with respect to lines of source code as the code is edited and/or verified.

In a first technique, the status information includes whether multiple copies of a statement have been generated. If a developer subsequently edits a source statement, the editor automatically determines whether copies of that statement exist, and automatically propagates the changes to each copy (preferably after prompting the developer and confirming an automatic propagate operation).

In a second technique, the status information includes whether the statement has previously been verified (e.g., in a code compilation process). If a developer subsequently copies the statement (e..g, by a cut and paste operation), the editor automatically determines whether the statement has previously been verified, and if not, automatically performs a responsive action (e.g., issuing a warning message).

Like many inventions in the realm of user interface, appellants' invention is not intended to make it possible to achieve an end result which was heretofore unachievable. It has always been possible to detect and correct multiple errors in code by manual methods. Appellants' invention is intended to make the development process easier, both by automatically alerting the developer before propagating potential errors and by automatically identifying copies and automatically editing them.

It will be observed that the ***automatic maintenance of status information*** by the development environment, and the ***automatic use of status to identify code statements*** for performing certain acts (such as warning of unverified status or prompting to propagate changes to copies of code statements), are critical features of appellants'

invention herein. A user could always do these things manually, or could manually invoke any of various search tools to find and replace particular errors. But if the user is compelled to take this action himself, he is likely to forget or to conclude that it is more trouble than it is worth, and not do it. Appellants' invention provides an improved interface because all these things are provided to the user automatically, without thought or effort on the user's part.

I. The Examiner improperly rejected independent claim1 (and dependent claims 3-6) under 35 U.S.C. §102(b) because *Miller* does not disclose key claim limitations, specifically automatically maintaining status of source code statements and using the status to identify copies for propagating changes

In order to support a rejection for anticipation, each and every element of the claimed invention must be shown in a single prior art reference. Appellants' claims are not anticipated by *Miller* because, inter alia, *Miller* fails to teach "automatically maintaining a record of status with respect to each source code statement", and using the status to automatically determine whether copies of a statement exist.

Miller discloses a pattern matching mechanism for editing text. *Miller*'s system is not specifically designed for editing source code, although it discloses that it could operate on, among other things, Java expressions (a form of source code). *Miller* proposes to create a extensible library of structure abstractions for text which can be reused. Various techniques are proposed in support of these ideas.

Miller's chapter 9 is cited in particular, dealing with selection inference. As disclosed in chapter 9, there are two techniques. In a first technique, called "selection guessing", a user inputs examples of what is desired, and the system infers a pattern to

make selections from the examples. In a second technique, called "simultaneous editing", the user manually specifies a group of records, and the system is constrained to make exactly one selection per record.

There are two basic flaws in the Examiner's read. First, pattern matching as described in *Miller* does not amount to maintaining "status" and identifying "copies", as recited in the claims. Second, *Miller*'s process requires manual input of pattern examples and/or specific records, and therefore does not meet the claim limitations of an automated process.

Appellants' claim 1 recites:

1. A method for developing source code for a computer program, comprising the steps of:

generating a plurality of source code statements in a source code file, said source code file being compilable into object code of said computer program;

automatically maintaining a record of status of each respective source code statement;

editing a first source code statement of said plurality of source code statements to produce an edited first source code statement;

automatically determining whether one or more copies of said first source code statement exist within said source code file from said status of each respective source code statement, each of said first source code statement and copy of said first source code statement occupying a different respective location within said source code file and being compilable together into said object code of said computer program; and

responsive to said automatically determining step, automatically propagating changes made by said editing step to said one or more copies of said first source code statement. [emphasis added]

A "copy" requires common derivation

The word "copy" implies common derivation, although not necessarily identity in all respects.[1] Mere similarity, no matter how close, does not make something a "copy" if there was no common derivation. A pattern matching mechanism such as disclosed in *Miller* has no capability to detect copying or copies. It can only detect similarities.

Appellants' claims recite maintaining "a record of status of each respective source code statement", and using this record to identify copies. The word "status" standing alone and without context, could mean many things. But in the context of the claim, it is used to identify a "copy", i.e. something having a common derivation with another. The only fair reading of this limitation is that the status information includes something which will enable one to identify common derivation.

The Examiner, in attempting to read claim 1 on *Miller*, apparently deems any intermediate data generated by the pattern matching mechanism to satisfy the required claim element of maintaining a record of status, and any entity identified by pattern-matching to be a "copy". Such a reading is clearly unreasonable. The pattern matching technique of *Miller* does not maintain status data with respect to individual records, and

---

[1] See, e.g., the following dictionary definitions of "copy"

1. An imitation, reproduction or transcript of an original. 2. one of the various examples or specimens of the same book, engraving or the like ... [Websters New Universal Unabridged Dictionary (Barnes & Noble Publishing 2003)]

1. an imitation or reproduction of something original; a duplicate. 2. One specimen or example of a printed text or picture ... [The American Heritage Dictionary of the English Language (Houghton Mifflin 1981)]

The first definition refers to something derived from a single original, while the second refers to something that is produced in multiple specimens, i.e. from the same source. Either implies common derivation. Even in the case of an "imitation", there is a conscious effort to create something similar to an original being copied. An object which is merely similar to another, without common derivation, is not a "copy".

does not identify copies, i.e., things having a common derivation. All *Miller*'s mechanism does is identify similar portions of text.

Pattern matching is not automated

But there is even a more serious flaw in the Examiner's proposed read. *Miller*'s pattern matching requires manual input from the user of pattern matching parameters, and does not meet the claim limitations of an automated process.

> *Miller*'s technique is summarized at the introduction to chapter 9 as follows:
>
> Chapter 7 described three ways to make selections in LAPIS – using the mouse, choosing a named pattern from the library pane, and writing a TC pattern. This chapter describes a fourth way – inferring selections from examples.
>
> Two techniques are described:
>
> - *Selection guessing* is the most general technique. It takes positive and negative examples from the user and infers a TC pattern consistent with the examples, which is then used to make the selection. At any time, the user can invoke an editing operation or a menu command on the current selection, start a fresh selection somewhere else, or tell the system to stop making inferences and add or remove selections manually with the mouse.
> - *Simultaneous editing* is a form of selection guessing specialized for a common case in repetitive text editing: applying a sequence of edits to a group of text regions. Simultaneous editing is a two-step process. The user first selects a group of records, such as lines or paragraphs or postal addresses. This record selection can be made like any other selection – using the mouse, writing a TC pattern, or using selection guessing. Once the desired records have been selected, the system enters a mode in which inference is constrained to produce exactly one selection in every record... [*Miller*, p. 221]

As explained above, a user either ***manually inputs*** multiple positive and negative examples, from which a pattern matching mechanism infers criteria for selecting records,

or the user ***manually specifies*** the records to be selected. Neither procedure meets the claim limitations.

Claim 1 recites a step of "automatically maintaining" status data. The Examiner apparently reads this on an automatic inferencing process, after the user has input the positive and negative examples for inferencing. The Examiner appears to be saying that any intermediate data generated during the pattern matching amounts to automatically maintaining status data. Such a reading effectively reads the limitation out of the claim.

Any computer implemented process generates intermediate state data. Clearly, this is not what is recited in the claim. The claim recites automatically maintaining status data in the context of manually editing source code. Clearly the step contemplates status data which is automatically generated and maintained while the user edits the source code, without any additional input from the user to maintain the status data.

As explained earlier, the very essence of appellants' invention is an improved user interface which does things without user intervention. When the user manually edits code, the system ***automatically*** generates the status data, from which it is later possible to determine whether there are copies of a statement. This is a very necessary requirement, for if the user were obligated to input the status data himself, or input search parameters or take similar action, very few users would bother. It has long been possible to search text and code strings for the occurrence of particular substrings. Any such search performed by a computer must necessarily generate intermediate state data. But these are essentially manual processes in the sense that search functions must be invoked and search parameters specified. They do not meet the requirements of claim 1.

For all the reasons specified above, *Miller* does not anticipate claim 1 (and claims dependent thereon), and the Examiner's rejections thereof were erroneous.

II. The Examiner improperly rejected independent claim 7 (and dependent claims 8 and 10-12) under 35 U.S.C. §102(b) because *Miller* does not disclose key claim limitations, specifically automatically maintaining status of source code and using the status to identify unverified source code statements.

Claim 7 recites a different aspect of the invention, i.e. the use of status data to detect unverified code statements and warning a developer before copying those statements. Like claim 1, claim 7 contains the limitation "automatically maintaining" status data, and the discussion above with respect to this limitation in claim 1 is applicable to claim 7 as well. For that reasons alone, the Examiner's rejection of claim 7 was erroneous. The additional limitations of claim 7 are discussed herein.

As is well known in the field of computer programming, source code for a computer program must conform to certain syntactical constraints of an applicable programming language. It must additionally provide a logical consistency to perform a desired function. When source code is subjected to a compilation process, the compiler generally verifies that the code satisfies the syntactical requirements of the language. Depending on the sophistication of the compiler, it may also verify certain requirements of logical consistency, although no compiler can detect all error in program logic. Other processes could conceivably be used instead of or in addition to conventional compilation to verify the correctness of source code.

If source code statements are copied from one code location to another before they have been subjected to a verification process (by compilation or otherwise), then there is

a risk that an error in the statement will be propagated to multiple locations. Appellants therefore propose to warn the user (or take other appropriate action) before the unverified code statement is copied. The user may then choose to verify the code statement before copying it, or take other appropriate action.

Appellants' claim 7 recites:

> 7. A method for developing source code for a computer program, comprising the steps of:
>
> generating a plurality of source code statements in a source code file, said source code file being compilable into object code of said computer program;
>
> automatically maintaining a record of status of each respective source code statement;
>
> receiving a user command to copy a first of said plurality of source code statements to a different location within said source code file to create a second source statement at said different location, said second source code statement being identical to said first source code statement, each of said first source code statement and said second source code statement being compilable together into said object code of said computer program;
>
> responsive to receiving said user command, *automatically determining whether said first source code statement has been previously verified from said status* of each respective source code statement; and
>
> if said first source code statement has not been previously verified, automatically performing at least one action in response to determining that said first source code statement is unverified. [emphasis added]

Miller, Chapter 10, discloses a pattern matching technique for identifying anomalous data points. I.e., applied to text structures, *Miller* discloses a technique whereby text structures which do not fit a pattern of previous examples are called to the attention of the user. In accordance with *Miller*, positive and negative examples of a pattern are provided by the user. The pattern could be anything; there is no explicit mention of source code verification (although a form of source code is mentioned earlier in the paper).

The Examiner's rejection, as nearly as appellants understand it, equates the pattern matching of data points with a verification process for source code statements. Appellants acknowledge that the claims are to be given their broadest reasonable interpretation, but submit that this reading is plainly unreasonable.

In the context of source code, verification is understood to mean some sort of process which confirms that the code complies with certain rules. It is deterministic. Pattern matching can only identify anomalies in data. Even if one ignores the fact that *Miller* does not disclose use of pattern matching for checking syntax of source code, pattern matching does not "verify" that data conforms to some set of rules. It only verifies that certain data points do or do not exhibit a pattern similar to others. To call this a "verification" is manifestly unreasonable.

But even if one concedes that which is unreasonable, the Examiner's read is still not made. For claim 7 recites the step of "responsive to receiving said user command [i.e., a command to copy a statement], automatically determining whether said first source code statement has been *previously verified* from said status..." Thus the claim recites that "verification" is done "previously", i.e. previous to receiving the user command to copy the statement. Even if *Miller*'s pattern matching is "verification", which it is not, *Miller* does not disclose that the results are somehow saved and later used to warn the user or take other action when the user attempts to copy a statement. Nor is there any disclosure of any automatic warning action taken responsive to a user's attempt to copy. As explained, Miller requires the user to manually input positive and negative examples.

Finally, although *Miller* does disclose elsewhere in his paper that his various searching and filtering methods might be used with a library of structure abstraction, such

as Java expressions, this oblique reference to a form of source code is not a disclosure that any of *Miller*'s techniques be used for source code verification, i.e. that source code meets the requirements of a language. It is simply inconceivable that one would apply a pattern matching technique for this purpose. Various deterministic source code verification mechanisms already exist in compilers, and pattern matching by example could hardly be more accurate or useful than conventional methods.

For all of the above reasons, claim 7 (and claims dependent thereon) are not anticipated by *Miller*, and the Examiner's rejections thereof were erroneous.

III. The Examiner improperly rejected the claims under 35 U.S.C. §103(a) because *Miller* is directed to entirely different subject matter, does not teach or suggest appellant's key features, and only with the added benefit of hindsight can key claim limitations be extracted from *Miller*.[2]

Claim 13 (and certain dependent claims) are program product claims analogous to claim 1 (and certain dependent claims). The Examiner rejected these claims under 35 U.S.C. §103(a) as obvious over *Miller*, using the same reasoning applied to claim 1 (and dependent claims) and the fact that it is well known to implement computerized processes as computer program products on computer-readable media. Appellants do not challenge this latter assertion, but challenge the reading of *Miller* on the method of claim 1 for the reasons explained above with respect to Point I of this Argument.

Appellants additionally make the following observations with respect to obviousness of all claims over *Miller*.

---

[2] Generally, a rejection for anticipation under 35 U.S.C. §102 may be deemed to include an implied or "subsumed" single reference rejection for obviousness under 35 U.S.C. §103. The subsumed obviousness rejection is addressed here.

The Examiner has constructed a very clever reading of isolated words and phrases in the various claims, but *Miller* does not disclose or suggest appellants' invention, when all the elements are read as a whole. Appellants claim a specific method for developing source code, which is aimed at a specific problem of errors being unintentionally multiplied when code sections are copied. Two specific techniques are claimed for a user interface which alleviates this particular problem. *Miller* relates to the generalized subject of pattern matching, which can be used for various operations on text. *Miller* does disclose that the it might be applied to, among other things, "Java expressions", a form of source code, but that is merely an observation of possible breadth of application. It does not amount to a specific technique for use with source code. *Miller* not only lacks a disclosure of the key elements of appellants' invention, but lacks any suggestion to apply his various techniques as claimed by appellants.

Even if one ignores all the inconsistencies and difficulties in adapting *Miller*'s pattern matching techniques to techniques such as claimed by appellants, one must ultimately ask where the suggestion to modify *Miller* in such a fashion comes from? It certainly doesn't come from *Miller*. The Examiner is reading *Miller* to be something it clearly is not, and proposing that *Miller*'s techniques be applied to something for which there is no indication they were intended. Appellants are entitled to ask, where in *Miller* is the suggestion to do all this? The Examiner is doing so on the basis of hindsight derived from appellants' disclosure, and nothing else.

For the reasons above stated, the claims are not obvious in view of *Miller*.

8. Summary

Appellants disclose and claim novel user interface techniques for developing computer code, wherein status of source statements is tracked as they are generated, edited and verified, and the status information is used either to identify statements which are copied (claims 1 and 13) or statements which a user proposes to copy but which are unverified (claim 7). These are specific user interface techniques intended to make it easier to develop code. *Miller* discloses generalized pattern matching and searching techniques which can be applied to, among other things, source code, but a disclosure or suggestion of the specific techniques claimed by appellants is lacking.

For all the reasons stated herein, the rejections for anticipation and obviousness were improper, and appellants respectfully request that the Examiner's rejections of the claims be reversed.

Date: May 1, 2007

Respectfully submitted,

CARY L. BATES, et al.

By_____

Roy W. Truelson, Attorney
Registration No. 34,265
(507) 202-8725 (Cell) (507) 289-6256 (Office)

From: IBM Corporation
Intellectual Property Law
Dept. 917, Bldg. 006-1
3605 Highway 52 North
Rochester, MN 55901

APPENDIX OF CLAIMS

1. A method for developing source code for a computer program, comprising the steps of:

generating a plurality of source code statements in a source code file, said source code file being compilable into object code of said computer program;

automatically maintaining a record of status of each respective source code statement;

editing a first source code statement of said plurality of source code statements to produce an edited first source code statement;

automatically determining whether one or more copies of said first source code statement exist within said source code file from said status of each respective source code statement, each of said first source code statement and copy of said first source code statement occupying a different respective location within said source code file and being compilable together into said object code of said computer program; and

responsive to said automatically determining step, automatically propagating changes made by said editing step to said one or more copies of said first source code statement.

2. (Cancelled)

3. The method for developing source code for a computer program of claim 1, wherein said automatically propagating step comprises:

automatically displaying said changes made by said editing step to at least one said copy of said first source code statement; and

soliciting user confirmation of said changes.

4. The method for developing source code for a computer program of claim 1, wherein said status of each respective source code statement comprises data indicating whether the respective source code statement has been verified.

5. The method for developing source code for a computer program of claim 4, wherein said data indicating whether a respective source code statement has been verified indicates whether the respective statement has been verified as part of a compilation process for compiling source code into object code executable by a computer system.

6. The method for developing source code for a computer program of claim 1, further comprising the steps of:

receiving a user command to copy a second of said plurality of source code statements to a different location within said source code file;

responsive to receiving said user command, automatically determining whether said second source code statement has been previously verified from said status of each respective source code statement; and

if said second source code statement has not been previously verified, automatically warning a user that said second source code statement is unverified.

7. A method for developing source code for a computer program, comprising the steps of:

generating a plurality of source code statements in a source code file, said source code file being compilable into object code of said computer program;

automatically maintaining a record of status of each respective source code statement;

receiving a user command to copy a first of said plurality of source code statements to a different location within said source code file to create a second source statement at said different location, said second source code statement being identical to said first source code statement, each of said first source code statement and said second source code statement being compilable together into said object code of said computer program;

responsive to receiving said user command, automatically determining whether said first source code statement has been previously verified from said status of each respective source code statement; and

if said first source code statement has not been previously verified, automatically performing at least one action in response to determining that said first source code statement is unverified.

8. The method for developing source code for a computer program of claim 7, wherein said step of automatically performing at least one action in response to determining that said first source code statement is unverified comprises issuing a warning message to a user.

9. (Cancelled)

10. The method for developing source code for a computer program of claim 7, wherein said step of automatically determining whether said first source code statement has been previously verified comprises automatically determining whether said first source code statement has successfully completed some portion of a compilation process for compiling source code into object code executable by a computer system.

11. The method for developing source code for a computer program of claim 7, wherein said status of each respective source code statement comprises data indicating whether the respective statement was copied from another source code statement.

12. The method for developing source code for a computer program of claim 11, wherein said step of automatically determining whether said first source code statement has been verified comprises automatically determining whether said first source code statement was copied from another statement which has been previously verified.

13. A computer program product for developing source code for a computer program, comprising:

a plurality of executable instructions recorded on tangible signal-bearing media, wherein said instructions, when executed by at least one processor of a digital computing device, cause the device to perform the steps of:

generating a plurality of source code statements in a source code file responsive to user input, said source code file being compilable into object code of said computer program;

automatically maintaining a record of status of each respective source code statement;

receiving a user input editing a first source code statement of said plurality of source code statements to produce an edited first source code statement;

automatically determining whether one or more copies of said first source code statement exist within said source code file from said status of each respective source code statement, each of said first source code statement and copy of said first source code statement occupying a different respective location within said source code file and being compilable together into said object code of said computer program; and

responsive to said automatically determining step, automatically propagating changes made by said editing step to said one or more copies of said first source code statement.

14. (Cancelled)

15. The computer program product for developing source code for a computer program of claim 13, wherein said automatically propagating step comprises:

automatically displaying said changes made by said editing step to at least one said copy of said first source code statement; and

soliciting user confirmation of said changes.

16. The computer program product for developing source code for a computer program of claim 13, wherein said status of each respective source code statement comprises data indicating whether the respective source code statement has been verified.

17. The computer program product for developing source code for a computer program of claim 16, wherein said data indicating whether a respective source code statement has been verified indicates whether the respective statement has been verified as part of a compilation process for compiling source code into object code executable by a computer system.

18. The computer program product for developing source code for a computer program of claim 13, wherein said instruction further cause the device to perform the steps of:

receiving a user command to copy a second of said plurality of source code statements to a different location within said source code file;

responsive to receiving said user command, automatically determining whether said second source code statement has been previously verified from said status of each respective source code statement; and

if said second source code statement has not been previously verified, automatically warning a user that said second source code statement is unverified.

APPENDIX OF EVIDENCE

No evidence is submitted.

APPENDIX OF RELATED PROCEEDINGS

There are no related proceedings.